Grafos – Aula 13

Roteiro

- Grafos com pesos negativos
- Caminho mais curto entre todos os pares
- Programação dinâmica
- Algortimo de Floyd-Warshall
- Melhorias

Distância em Grafos

- **Problema**: Dado G com pesos, determinar menor caminho entre dois vértices

Dijkstra!

- **Problema**: Menor caminho entre *todos os pares* de vértices?

Dijkstra n vezes!

Pesos Negativos

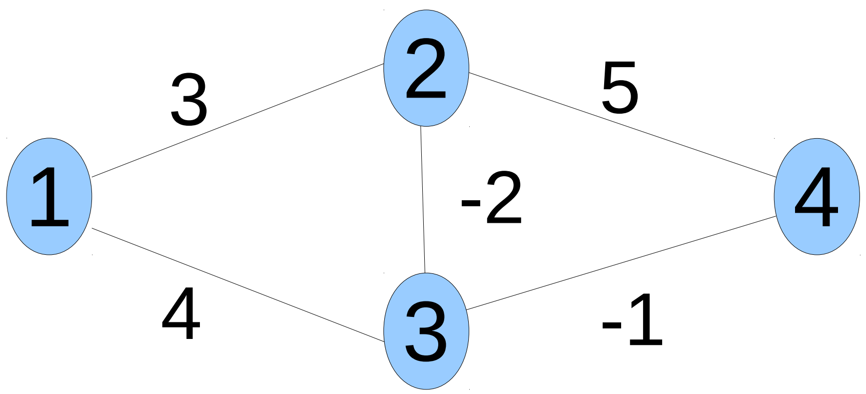
- **Limitação**: Dijkstra funciona apenas para grafos com pesos positivos nas arestas

Por que Dijkstra falha?

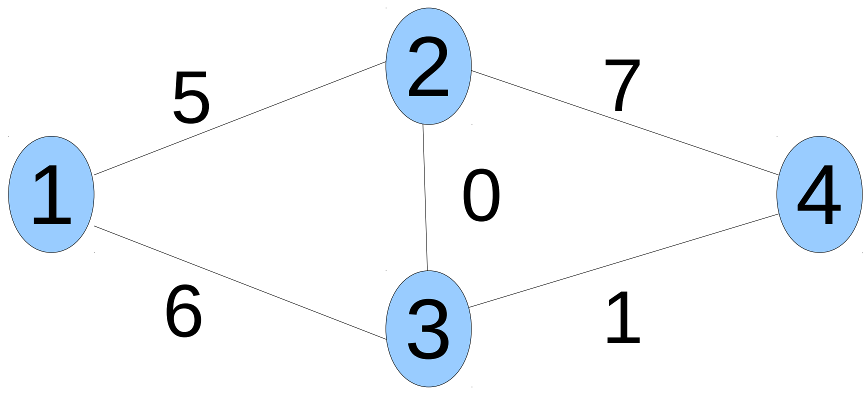
- Guloso é míope: não considera o futuro, e não revisita vértices já explorados

- **Ideias:** transformar o grafo
  - somar o módulo do menor valor negativo em todos os pesos, funciona?

Exemplo



- Menores caminhos (simples)
  - entre 1 e 2: (1,3,2), custo 2
  - entre 1 e 3: (1,2,3), custo 1
  - entre 1 e 4: (1,2,3,4), custo 0
- Dijkstra no vértice 1
  - entre 1 e 2: (1,2), custo 3
- Somar |-2| em todas as arestas



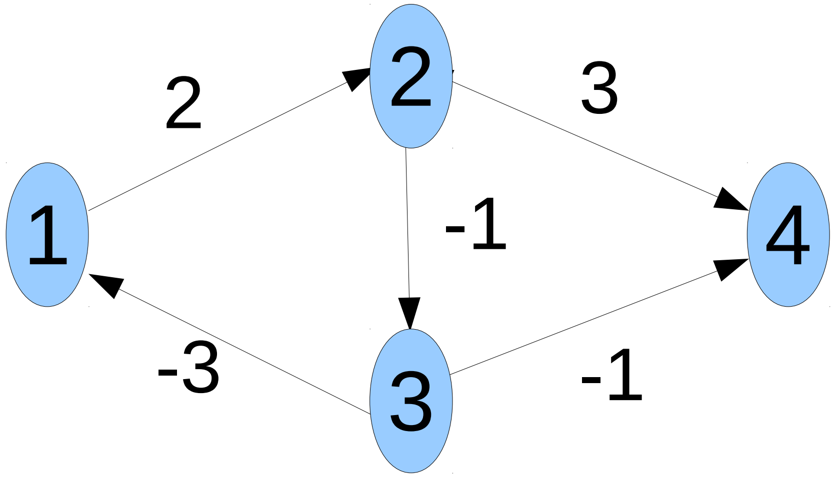
- Dijkstra no vértice 1
  - entre 1 e 2: (1,2), custo 5
  - não funcionou, *guloso é miope*

Pesos Negativos

- Grafos com pesos negativos surgem na prática
- Transações financeiras entre ativos (perder ou ganhar)
- Viagem de motoristas em estradas com pedágios (ganha e perde)
- Modelagem de reações químicas e calor (libera e demanda)

Ciclos Negativos

- Ciclos em grafos com pesos negativos



- Custo do menor caminho (não simples) entre 1 e 4?
- **Indefinido!**
- Caminho mais curto definido apenas quando não há ciclos negativos

Distância em Grafos

- Considerar grafo com pesos negativos, distância é soma dos pesos
- **Problema 1**: calcular distância entre todos os pares de vértices
- **Problema 2**: calcular caminho mínimo entre todos os pares de vértices
- **Problema 3**: detectar ciclos negativos
  - no final, assumir isto no momento

Algoritmo para estes problemas?

Programação Dinâmica

- Técnica para construção de algoritmos
- Baseada na análise e decomposição da estrutura do problema
- Recursão da solução ótima
- Memorização (armazenamento) de resultados temporários

Técnica fundamental e poderosa!

- Geralmente difícil ser empregada em casos gerais

Caminho Mínimo

- Considere um par de vértices i, j

O que é a solução ótima?

- Caminho mais curto de i até j
  - $p = (i, v_1, v_2, ..., j)$
  - $v_1, v_2, ...$: vértices intermediários
  - i e j não podem ser vértices intermediários

O que podemos dizer sobre *p*?

- Considere o último vértice do grafo, ou seja, vértice *n*
- Ou *n* pertence a *p* ou *n* não pertence a *p*

Análise do Caminho Mínimo

- Se *n* não pertence a *p*

O que podemos afirmar?

- p = (i, v_1, v_2, ..., j) é caminho mínimo também para o grafo sem vértice n

- Se *n* pertence a *p*

O que podemos afirmar?

- p = (i, ..., n , ..., j) é caminho mínimo
- p_1 = (i, ..., n) e p_2 = (n, ..., j) são caminhos mínimos

Análise do Caminho Mínimo

- Se p_1 = (i, ..., n) e p_2 = (n, ..., j) são caminhos mínimos

O que podemos afirmar sobre distâncias?

- d(i, j) = distância entre i e j?
- d(i, j) = d(i, n) + d(n, j)
- Além disso:
- p_1 = (i, ..., n) é mínimo usando como vértices intermediários 1, ..., n-1
- p_2 = (n, ..., j) é mínimo usando como vértices intermediários 1, ..., n-1

Construindo uma Recursão

- p_1 = (i, ..., n) e p_2 = (n, ..., j) são caminhos mínimos sem utilizar *n* como intermediário
- Problema menor, com um vértice a menos!

Como construir recursão?

- Usar variável para indicar quais vértices intermediários estão sendo considerados na construção do caminho mínimo
- d(i,j,k) : distância entre i e j quando consideramos os vértices 1,..., k como intermediários
- d(i,j,n) é valor final, distância entre i e j no grafo (considera todos os vértices)

Construindo uma Recursão

- Se vértice n não pertence ao caminho mínimo, então temos
  - d(i,j,n) = d(i,j,n-1)
- Se vértice n pertence ao caminho mínimo, então temos
  - d(i,j,n) = d(i,n,n-1) + d(n,j,n-1)

- Das duas opções, queremos a menor
- d(i,j,n) = min (d(i,j,n-1), d(i,n,n-1) + d(n,j,n-1))

Generalizando

- Considere o conjunto de vértices intermediários {1,2, ..., k}.
- Considere o par de vértices i, j
- Considere a solução ótima (distância) usando este conjunto intermediário
- Se vértice k não pertence ao ótimo, então temos
  - $d(i,j,k) = d(i,j,k-1)$
- Se vértice k pertence ao ótimo, então temos
  - $d(i,j,k) = d(i,k,k-1) + d(k,j,k-1)$
- Logo, temos
  - $d(i,j,k) = \min(\, d(i,j,k-1),\ d(i,k,k-1) + d(k,j,k-1)\,)$

Algoritmo

- Algoritmo iterativo para calcular distâncias

```
Floyd-Warshall(A)
   Array d[1,...,n ; 1,...,n; 0,...,n]
   d[i,j,0] = A[i,j]; // peso das arestas
   d[i,i,0] = 0;      // peso zero

   for k = 1, ..., n
      for i = 1, ..., n
         for j = 1, ..., n
            d[i,j,k] = min(d[i,j,k-1] ,
                           d[i,k,k-1] + d[k,j,k-1] )
   return d[*,*,n]   // matriz de distâncias
```

- Complexidade?
  - memória e tempo de execução: $\Theta(n^3)$

Algoritmo de Floyd-Warshall

- Descoberto por Floyd e Warshall em 1962 de maneira independente
- Determina distância mínima entre todos os pares de vértices de um grafo
  - Complexidade $\Theta(n^3)$
- **Impressionante:** grafo pode ter $\Theta(n^2)$ arestas e *todos* os caminhos são considerados entre *todos* os $\Theta(n^2)$ pares de vértices

Poder de fogo da Programação Dinâmica!

Algoritmo Melhorado

- Como reduzir uso de memória para $\Theta(n^2)$?
- Manter apenas 1 matriz de distâncias, atualizar na própria matriz

```
Floyd-Warshall(A)
   Array d[1,...,n; 1,...,n]
   d[i,j] = A[i,j]; // peso das arestas
   d[i,i] = 0;      // peso zero

   for k = 1, ..., n
      for i = 1, ..., n
          for j = 1, ..., n
            d[i,j] = min(d[i,j], d[i,k]+d[k,j])
   return d;
```

- Se convencer que nada (contas) mudou!

Detectando Ciclos Negativos

Como detectar ciclos negativos?

- **Idéia**: se grafo tem ciclo negativo, custo para ir do vértice a ele mesmo é menor do que zero!
- Algoritmo atualiza todas as distâncias, inclusive entre o par de vértices i, i
- d(i,i) é considerada a cada passo k
- se $d(i,k) + d(k,i) < 0$, grafo tem ciclo negativo
- Ao final do algoritmo, se $d(i,i) < 0$ para algum i, então temos ao menos um ciclo negativo

Mantendo Menor Caminho

Como obter o menor caminho?

- **Idéia**: manter matriz de pais para cada par de vértices i (orgem), j (destino)
  - pred(i, j) : pai do vértice j no caminho mínimo de i para j
- Inicializar com a matriz de adjacência
- Atualizar toda vez que distância entre i e j for atualizado passando por vértice k
  - pred(i, j) = pred(k, j)
- Ao final, matriz pred(i,j) codifica todos os caminhos mínimos!